PROGRAMMATION
en langage BASIC

Le BASIC que nous allons utiliser est le **PicBASIC** élaboré par la Société ***M***ICRO ***E***NGINEERING ***L***abs (représentée en France par SELECTRONIC à Lille).

Le **COMPILATEUR PicBASIC** tient sur une disquette.
Il est proposé en deux versions : la version standard et la version professionnelle.
Tous les exemples que je fournis ont été réalisés avec la version standard, qui permet de réaliser n'importe quel programme, sans limitations.
Ce que l'on trouve en plus dans la version Pro, c'est un plus grand confort de travail : possibilité d'ouvrir plusieurs fenêtres en même temps, numérotation des lignes du fichier source, etc..
La disquette est accompagnée d'un manuel d'utilisation en anglais et d'une traduction en français.
Ce logiciel, vendu par **SELECTRONIC** à Lille, au moment de la rédaction de ces pages, coûte € 129,50 (prix au 16/09/2002).

Si vous voulez mon conseil : n'hésitez pas à vous le procurer.

Il fera de vous des programmeurs enthousiastes, car programmer avec le PicBASIC de MICRO ENGINEERING Labs (MEL) est extrêmement facile.

Il n'y a rien de comparable entre la programmation en langage ASSEMBLEUR et la programmation en langage MEL PicBASIC.

Bien qu'il existe d'autres BASIC pour les microcontrôleurs PIC, proposés par d'autres Sociétés, celui de MEL est de loin le meilleur, car extrêmement puissant et très facile à prendre en main.

Alors un conseil : ne cédez pas à la tentation de vous en procurer ou d'accepter une version quelconque de BASIC pour PIC, pensant faire une bonne affaire. Car vous vous habitueriez à travailler avec ses instructions et – si un jour vous vous rendiez compte des limites de votre BASIC et vous vouliez passer au MEL PicBASIC – vous auriez de la difficulté à chasser de votre mémoire les instructions de l'ancien BASIC, avec le risque de les mélanger et fatalement d'avoir à corriger des erreurs.

Faites donc dès à présent l'effort de cet achat, et je vous assure que vous ne le regretterez pas. Le langage ASSEMBLEUR, l'architecture du

PIC, la Page 0, la Page1, le Registre STATUS, le Registre OPTION et tout le reste.. sera vite oublié. Vous ne verrez plus que votre programme.

C'est comme si, pour préparer un programme commandant l'ouverture de la porte de votre garage :

a) avec le langage ASSEMBLEUR vous devriez écrire :
- approchez-vous de la porte du garage
- mettez la main gauche dans la poche gauche de votre veste
- prenez la clé de la porte du garage
- avez-vous la clé en main ?
- non
- mettez alors la main droite dans la poche droite de votre veste
- prenez la clé de la porte du garage
- avez-vous la clé en main ?
- oui
- sortez la main droite de la poche droite de votre veste
- introduisez la clé dans la serrure de la porte du garage
- tournez la clé
-etc...
- etc...
- fin

b) alors qu'avec le langage MEL PicBASIC il suffirait d'écrire :
- approchez-vous de la porte du garage
- prenez la clé
- ouvrez la porte du garage
- fin

Ce sont les instructions du PicBASIC qui - avec leur puissance - comprennent ce que vous voulez faire et effectuent toutes les petites tâches intermédiaires, à votre place, sans que vous ayez besoin de les détailler.

Merveilleux, non ?

Avant d'examiner chaque instruction une par une, il est indispensable de s'arrêter sur quelques particularités qui facilitent grandement l'apprentissage du langage MEL PicBASIC.

Symbol

Avec la directive Symbol on peut associer un nom à une variable ou à une pin du microcontrôleur.

Ainsi définies, telle variable ou telle pin du microcontrôleur peuvent alors être plus facilement utilisées dans l'écriture d'un programme.

Par exemple : Symbol LED = Pin 0
Signifie : associe à *Pin 0* (c'est à dire à RB0) le nom *LED*.
Ainsi, par exemple l'instruction :

LED = 1

est plus facile à lire dans le programme, car on comprend qu'elle allume la LED (met au niveau logique haut la pin 0 du Port B).

Symbol permet de renommer un chiffre (le plus souvent) ou une variable.
Chiffres et variables qui – rencontrés dans la lecture d'un programme – ne diraient pas grand'chose, deviennent plus explicites après avoir été *rebaptisés* par Symbol car il est beaucoup plus facile de retenir un nom qu'une adresse, le nom étant chargé d'une signification, alors qu'une adresse ne dit rien.

Les variables de type B et W

Les variables correspondent à des emplacements mémoire utilisés pour stocker temporairement des données.

Le MEL PicBASIC propose d'emblée des variables prêtes à l'emploi.

Selon que les données à stocker soient de petites données et qu'elles tiennent en un seul octet (*byte* = 8 bits), ou qu'elles soient de grandes données et que pour les accueillir il faille un mot (*word* = 16 bits), MEL PicBASIC propose deux types de variables :

a) les **variables de type B** (*byte*) utilisées pour stocker des variables à 8 bits, et

b) les **variables de type W** (*word*) utilisées pour stocker des variables à 16 bits.

Pour le microcontrôleur 16F84 PicBASIC a prédéfini 52 variables de type B (**de B0 à B51**), et 26 variables de type W (**de W0 à W25**).

Les variables de type W sont constituées de la juxtaposition de deux variables de type B. Ainsi la variable W0 est constituée de la juxtaposition de la variable B0 et de la variable B1. La variable W1 est constituée de la juxtaposition de la variable B2 et de variable B3, et ainsi de suite..

Le compilateur ne peut utiliser que les registres du microcontrôleur pour stocker les variables du programme. Ce qui veut dire que le nombre des variables mises à disposition dépend du modèle de PIC utilisé.

Les variables de bit

Il faut savoir qu'il existe aussi des variables de bit prédéfinies.
Ce sont les variables B0 et B1 qui les accueillent.
Dans ce cas, les variables de bit qu'elles logent s'appellent :

Bit0, **Bit1**, **Bit2** .... et ainsi de suite jusqu'à **Bit15**.

Dénomination des lignes de PORTS

MEL PicBASIC n'appelle pas de la même façon les bits du Port A et les bits du Port B.

a) **Les bits du Port A**
Les bits du Port A sont appelés **: Pin0**
Pin1
Pin2
Pin3
Pin5

b) **Les bits du Port B**
Les bits du Port B sont appelés **: 0**
1
2
3
4
5
6
7

Attention donc !

- Pour vous référer à l'un des 5 bits du Port A vous direz : Pin0, Pin1, Pin2, Pin3, Pin4.

- Pour vous référer à l'un des 8 bits du Port B vous direz : 0, 1, 2, 3, 4, 5, 6, 7.

Configuration des bits de chaque Port

Chaque bit de Port (Port A et Port B) peut être configuré en entrée ou en sortie :

- tout bit mis à 0 est configuré en *entrée*
- tout bit mis à 1 est configuré en *sortie*.

Mais la façon dont MEL PicBASIC configure les bits n'est pas la même pour le Port A et pour le Port B.

Port A

a) **<u>Pour configurer le Port A en *entrée*</u>** (associé - par exemple - à des interrupteurs) il faut premièrement s'adresser au Registre du Port A (situé à l'adresse 5). Puis il faut définir la direction de ce Port en s'adressant au Registre TRISA (situé à l'adresse 85).

Les bits mis à 0 deviennent des *entrées*. Tandis que les bits mis à 1 deviennent des *sorties*.

Pour configurer tout le Port A en *entrée* il faut donc écrire :

```
Symbol  PORTA = 5
Symbol  TRISA  = $85
POKE    TRISA,255          (255 = 11111111)
```

(Tous bits du Port A en *entrée*).

b) **<u>Pour configurer le Port A en *sortie*</u>** il faut faire de même : premièrement s'adresser au Registre du Port A (situé à l'adresse 5). Puis définir la direction de ce Port en s'adressant au Registre TRISA (situé à l'adresse 85).

Les bits mis à 0 deviennent des *entrées*. Tandis que les bits mis à 1 deviennent des *sorties*.

Pour configurer tout le Port A en *sortie* il faut donc écrire :

```
Symbol  PORTA = 5
Symbol  TRISA = $85
POKE    TRISA,0            (0 = 00000000)
```

(Tous bits du Port A en *sortie*).

Port B

<u>Pour configurer le Port B</u> on utilise l'instruction

DIRS = X X X X X X X X

DIRS suivi du signe égal et - après le signe égal - des 0 et des 1 formant *l'octet de configuration*, conformément à la façon dont on veut configurer le Port.

0 = *entrée*
1 = *sortie*

Exemples :

- DIRS = % 11111111 (ou DIRS = 255) — (Tous bits du Port B configurés en *sortie*).

- DIRS = % 00000000 (ou DIRS = 0) — (Tous bits du Port B configurés en *entrée*).

- DIRS = % 00001111 (ou DIRS = 15) — (Bits 0 à 3 configurés en *sortie, et* bits 4 à 7 configurés en *entrée*)

- etc.. (tous les cas de figure sont permis).

Lecture des Ports

<u>Pour lire l'état d'un Port</u> on utilise l'instruction PEEK.
Comme ceci :

PEEK PORTA,B0

Ce qui signifie : lire le contenu du Port A et le stocker dans la variable B0. En fait cette variable peut être l'une quelconque des variables prédéfinies pour le 16F84 (B0... B51).

Ensuite on peut affiner par :

IF B0, Bit0 = 0 Then......

ou

IF B0, Bit0 = 1 Then.....

etc..

Les constantes

Les constantes numériques peuvent être exprimées en trois bases :

- décimale
- binaire
- hexadécimale

Pour dire au compilateur en quelle base on veut travailler, on place un préfixe devant le nombre.

En écrivant un nombre **sans préfixe**, le compilateur traite ce nombre en *décimal*.
Par exemple : 100 (sans préfixe) est interprété comme 100 (base décimale).

En tapant **%100**, le compilateur interprète le nombre 100 en *binaire* (00000100 c'est à dire 4 en décimal).

Enfin, en tapant **$100**, le compilateur interprète le nombre 100 en *hexadécimal* (0001 0000 0000 c'est à dire 256 en décimal).

Les labels

Les labels (étiquettes) doivent obligatoirement commencer à la première colonne, et doivent se terminer par deux points (:)

Elles servent généralement à identifier des sous-programmes pour que le PIC - à l'issue d'un test, par exemple - fasse une certaine chose si la condition est vraie, ou une autre chose si la condition est fausse. Ou, devant la position d'un interrupteur, pour qu'il fasse une certaine chose si l'interrupteur est ouvert ou une certaine autre chose si l'interrupteur est fermé.... etc..

Les étiquettes repèrent des adresses, pour que le PIC y accède quand on y fait référence.

Les commentaires

En PicBASIC les commentaires doivent obligatoirement être précédés par une apostrophe (‘) (Attention : la directive REM n’existe pas en PicBASIC).

Tout ce qui est écrit après une apostrophe est ignoré par le compilateur.

Les commentaires servent à garder une trace de ce que fait le programme.

L’écriture d’un programme sans commentaires n’est pas à envisager, car même si la compréhension d’un programme peut paraître évidente au moment de sa création, elle peut s’avérer difficile quelque temps après, ou sa lecture impossible pour ceux qui devraient en assurer la maintenance.

Ne soyez pas avares de commentaires. Mettez-les même quand les seules instructions du programme pourraient donner une impression d’évidence.

Le set d'INSTRUCTIONS du MEL PicBASIC

Le MEL PicBASIC est constitué de 40 instructions.

Ne confondez pas - je vous prie - les 40 instructions de ce langage avec les 37 instructions du langage ASSEMBLEUR : cela n'a strictement rien à voir !

Examinons ces 40 instructions une par une, en les présentant par ordre alphabétique pour que vous les retrouviez plus facilement.

Je signale qu'elles peuvent être écrites comme on veut :
- en majuscules
- en minuscules
- et même en mélangeant majuscules et minuscules.

Par exemple, l'instruction GOTO peut s'écrire :

- GOTO
- Goto
- goto
- GoTO
- goTO
- gOto
- etc...

Classement par ordre alphabétique

BRANCH
BUTTON
CALL
DEBUG
DIRS =
EEPROM
END
FOR/NEXT
GOSUB
GOTO
HIGH
I2Cin
I2Cout
IF...THEN
INPUT
LET
LOOKdown
LOOKup
LOW
NAP
OUTPUT
PAUSE
PEEK
PINS =
POKE
POT
PULSin
PULSout
PWM
RANDOM
READ
RETURN
REVERSE
SERin
SERout
SLEEP
SOUND
TOGLE
TRISA
WRITE

Classement par genre

BRANCH CALL GOSUB GOTO RETURN	BUTTON DIRS = HIGH INPUT LOW OUTPUT PINS = POT PULSin PULSout PWM REVERSE SERin SERout SOUND TOGLE	I2Cin I2Cout TRISA	DEBUG EEPROM END FOR...NEXT IF...THEN LET LOOKDOWN LOOKUP NAP PAUSE PEEK POKE RANDOM READ SLEEP WRITE

BRANCH

Cette instruction est une variante de GOTO.
Comme GOTO, elle permet de sauter à des sous-programmes.
Mais ici en fonction de la valeur prise par la variable donnée en offset.

SYNTAXE :
BRANCH *offset* (Label1, Label2, ...)
Si *offset* vaut 0, le programme saute à Label1
S'il vaut 1, le programme saute à Label2
et ainsi de suite..

EXEMPLE :
BRANCH B5 (Label1, Label2, Label3)
Si B5 vaut 0, le programme fait un GOTO au sous-programme appelé Label1.
S'il vaut1, le programme fait un GOTO au sous-programme appelé Label2.
S'il vaut2, le programme fait un GOTO au sous-programme appelé Label3.

Si la valeur de l'*offset* (ici B5) prend une valeur supérieure au nombre des sous-programmes indiqués dans les parenthèses (ici : 0, 1, 2) le programme n'effectue aucun saut et continue en séquence.

BUTTON

Cette instruction lit l'état d'un interrupteur placé sur l'une des 8 lignes (0.....7) du Port B (RB) et peut :

- déclencher une procédure d'anti-rebonds
- déclencher une procédure d'auto-répétition
- tester si l'interrupteur ferme vers la masse ou vers le +
- faire un GOTO à un sous-programme, selon que l'interrupteur soit ouvert ou fermé.

SYNTAXE :
BUTTON *Pin, Down, Delay, Rate, Var, Action, Label*

Label : Nom du sous-programme auquel il faut sauter.

Action : Etat auquel l'interrupteur doit se trouver (0 si ouvert, 1 si fermé) pour que le saut ait lieu.

Var : Nom de la variable (B0....B51) servant au décomptage des auto-répétitions.

Rate : Nombre de fois (0.....255) qu'on veut faire tourner l'auto-répétition pour lire l'état de l'interrupteur.

Delay : Nombre de cycles (0.....255) qu'on veut laisser passer à partir du moment où l'interrupteur a été actionné, avant de déclencher une procédure d'auto-répétition. Si *Delay* vaut 0, les fonctions anti-rebonds et répétition automatique n'ont pas lieu. Si *Delay* vaut 255, seule la fonction anti-rebonds a lieu, pas celle d'auto-répétition.

Down : Etat de l'interrupteur au moment où il est appuyé.

Pin : Numéro de la pin du Port B (RB) sur laquelle se trouve l'interrupteur (0.......7).

EXEMPLE :
BUTTON 2, 0, 100, 10, B0, 0, CLIGNOTE
Lit l'état de l'interrupteur placé sur RB2 (en supprimant les rebonds des contacts) et – si l'interrupteur est ouvert – saute au sous-programme appelé CLIGNOTE.

CALL

Appelle un sous-programme écrit en langage ASSEMBLEUR.

C'est une instruction qui assure une sorte de passerelle, grâce à laquelle on peut incorporer, à des instructions écrites en PicBASIC, des instructions écrites en langage ASSEMBLEUR.

SYNTAXE :
CALL *Label*

EXEMPLE :

```
CALL   DELAI
DELAI  DECFSZ     COMPT1,1
       GOTO       DELAI
       MOVLW      .255
       MOVWF      COMPT1

       DECFSZ     COMPT2,1
       GOTO       DELAI
       MOVLW      .255
       MOVWF      COMPT1
       MOVLW      .255
       MOVWF      COMPT2
       RETURN
```

DEBUG

Sert lors de la mise au point d’un programme.

La mise au point d’un programme est une phase délicate.

Le programme est écrit, mais des erreurs de syntaxe ou de compilation peuvent apparaître. Le programme ne se déroule pas correctement. Il faut alors le *déboguer*.

Pour cela il existe différentes techniques.

L’une d’elles consiste à placer une instruction DEBUG dans le programme pour provoquer l’affichage de valeurs permettant de délimiter des étapes dans le déroulement du programme et localiser l’erreur.

L’instruction DEBUG se comporte comme une sorte de point d’arrêt dynamique. Elle ne peut être utilisée que pendant la phase du développement, et ne peut pas figurer dans un programme mis au point.

Dans tous les cas, il faut retirer cette instruction d’aide à la mise au point, lors de la fabrication de la version finale du programme.

De toutes façons, si l’instruction DEBUG restait dans le programme, elle serait ignorée une fois que celui-ci ait été correctement compilé.

Au fur et à mesure que vous réalisez votre programme, ayez toujours la mise au point en tête.

Si vous modifiez temporairement des instructions, ayez une feuille volante avec leur liste. C’est la meilleure façon pour ne pas oublier de toutes les restaurer ensuite.

DIRS

Définit la direction des lignes du Port B.

SYNTAXE :
DIRS = X X X X X X X X

(octet de configuration)
(0.......255)

0 = *entrée*
1 = *sortie*

EXEMPLES :
1)
DIRS = 255 (255 décimal, donc 11111111 en binaire)
Cela peut s'écrire aussi :
DIRS = %11111111
Tous bits du Port B configurés en sortie

2)
DIRS = 0
Tous bits du Ports B configurés en entrée.

EEPROM

C'est l'instruction avec laquelle on écrit dans l'EEPROM du PIC, dans des adresses consécutives, à partir de celle qu'on lui indique comme étant l'adresse de départ.

SYNTAXE :
EEPROM *Adresse*, (*Donnée, Donnée, Donnée,......)*

Dans les parenthèses - espacées par des virgules - les données à écrire dans l'EEPROM (constantes numériques ou caractères ASCII).

Adresse de départ, à partir de laquelle on veut écrire dans l'EEPROM.
Si on ne met rien dans ce paramètre, l'écriture démarre automatiquement à l'adresse 0.
(NB : Dans le µC 16F84 l'étendue de l'EEPROM va de 00 à 3F).

EXEMPLE :
EEPROM 8, (2, 4, 8, 16, 32, 64, 128)

On stocke 2 à l'adresse EEPROM 08
On stocke 4 à l'adresse EEPROM 09
On stocke 8 à l'adresse EEPROM 0A
On stocke 16 à l'adresse EEPROM 0B
On stocke 32 à l'adresse EEPROM 0C
On stocke 64 à l'adresse EEPROM 0D
On stocke 128 à l'adresse EEPROM 0E

NB : Les données entrent dans la mémoire EEPROM une seule fois : lors de la programmation du circuit, et non pas chaque fois qu'on exécute le programme.

END

C’est l’instruction que l’on doit obligatoirement placer à la fin de tout programme.

Elle termine l’exécution du programme et place le microcontrôleur en mode veille.

Pour réveiller le microcontrôleur on est obligé de faire un Reset matériel.

FOR... NEXT

Cette instruction introduit une boucle en faisant exécuter un certain nombre de fois les actions détaillées dans ce que l'on appelle : le *corps de l'instruction*.

SYNTAXE :
FOR *Variable* = *Début* TO *Fin* (*Pas*)
.
.
. .
.
NEXT *Variable*

Variable prend d'abord la valeur définie par *Début ;* puis – successivement – toutes les valeurs suivantes, par incrémentation automatique, jusqu'à atteindre la valeur définie par *Fin*, et chaque fois, pour chaque valeur de *Variable*, exécute les instructions détaillées dans le *corps de l'instruction*.

A chaque passage, *Variable* s'incrémente (de 1).

Mais on peut aussi incrémenter la *Variable* au pas qu'on veut. Dans ce cas il faut le préciser en donnant une valeur à *Pas*, placée entre deux parenthèses.

Par exemple, si l'on écrit :
FOR B1 = 0 TO 10 (2)
B1 prendra successivement
non pas les valeurs 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9
mais 0, 2, 4, 6, 8 et 10.

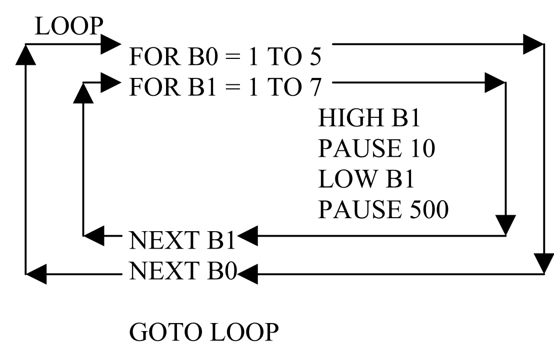
EXEMPLE :

```
FOR B1 = 1 TO 10
        HIGH 0
        PAUSE 200
        LOW 0
        PAUSE 200
NEXT B1
```

Fait clignoter *10 fois de suite* (allume pendant 200 millisecondes, puis éteint pendant 200 millisecondes) la LED connectée sur RB0 (Pin 0 du Port B).

NB : Des boucles FOR... NEXT peuvent être imbriquées dans d'autres boucles FOR... NEXT jusqu'à un maximum de 16 imbriquements.

EXEMPLE :



GOTO LOOP

Pour chacune des valeurs pouvant être prises par B1 (c'est à dire 1 à 7. C'est la *boucle interne*) faire le travail défini par la *boucle interne* autant de fois que les valeurs pouvant être prises par B0 (*boucle externe*).

Dans le cas de cet exemple : la boucle externe tourne 5 fois, avant que B1 passe d'une valeur à la suivante.

Ici, donc, la LED reliée à RB0 clignote 5 fois, puis la LED reliée à RB1 clignote 5 fois, puis la LED reliée à RB2 clignote 5 fois.... jusqu'à celle reliée à RB7.

Puis, ça recommence.

Restons sur cet exemple, et étudions-le de plus près.
Cela en vaut la peine, car les boucles FOR.... NEXT sont très utilisées quand on programme en PicBASIC.

```
LOOP
        FOR B0 = 1 TO 5
        FOR B1 = 0 TO 7
                HIGH B1          ⎫
                PAUSE 10         ⎬  Le travail...
                LOW B1           ⎪  que doivent faire
                PAUSE 500        ⎭  chacune des valeurs
        NEXT B1 <------------------ de B1, autant de fois...
        NEXT B0 <-------- .... que chacune des valeurs de B0

        GOTO LOOP
```

Autrement dit :

```
HIGH 0: PAUSE 10: LOW 0 : PAUSE 500
HIGH 0: PAUSE 10: LOW 0 : PAUSE 500
HIGH 0: PAUSE 10: LOW 0 : PAUSE 500
HIGH 0: PAUSE 10: LOW 0 : PAUSE 500
HIGH 0: PAUSE 10: LOW 0 : PAUSE 500

HIGH 1: PAUSE 10: LOW 1 : PAUSE 500
HIGH 1: PAUSE 10: LOW 1 : PAUSE 500
HIGH 1: PAUSE 10: LOW 1 : PAUSE 500
HIGH 1: PAUSE 10: LOW 1 : PAUSE 500
HIGH 1: PAUSE 10: LOW 1 : PAUSE 500


HIGH 2: PAUSE 10: LOW 2 : PAUSE 500
```

etc....
jusqu'à 7.

L'importance de NEXT est très grande. Voyons-la au moyen d'un autre exemple :

On veut allumer toutes les LED du Port B.
Ecrivons le programme suivant :

```
LOOP    FOR B0 = 0 TO 7
                HIGH B0
                NEXT B0
END
```

Si on exécute ce même programme sans l'instruction NEXT B0, on constate que seule la LED reliée à RB0 s'allume, et pas les autres.

Ceci parce que l'instruction NEXT B0 incrémente la variable B0 et referme la boucle en fournissant – à chaque tour – une nouvelle valeur (0, 1, 2,.....7) correspondant à différents RB et – par là – à des LED différentes.

GOSUB

C'est l'instruction par laquelle on appelle un sous-programme.

Le sous-programme appelé doit impérativement se terminer par l'instruction RETURN.

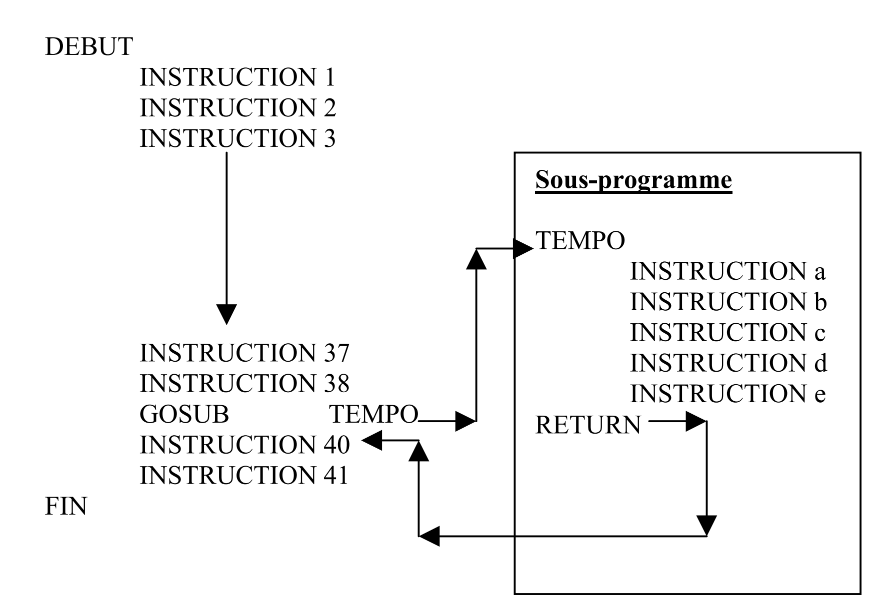
L'instruction RETURN, placée à la fin du sous-programme, fait revenir le programme principal à l'instruction se trouvant juste après GOSUB.

Ne pas confondre GOSUB avec GOTO qui renvoie – elle aussi – l'exécution du programme ailleurs, mais – cette fois – sans retour.

SYNTAXE :
GOSUB *Label*
.
.
.
RETURN

NB : Des paires d'instructions GOSUB..... RETURN peuvent être imbriquées. Mais sans aller au-delà de 4 niveaux (on ne peut pas imbriquer plus de 4 paires l'une dans l'autre).

L'exemple qui suit, illustre le fonctionnement de GOSUB et de RETURN.


Programme principal
DEBUT
INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
INSTRUCTION 37
INSTRUCTION 38
GOSUB TEMPO
INSTRUCTION 40
INSTRUCTION 41
FIN
Sous-programme
TEMPO
INSTRUCTION a
INSTRUCTION b
INSTRUCTION c
INSTRUCTION d
INSTRUCTION e
RETURN

GOTO

Renvoie à un autre endroit du programme.

SYNTAXE :
GOTO *Label*

↓

Repère (adresse) de l'endroit de la mémoire auquel le programme doit se rendre pour poursuivre son exécution.

EXEMPLE :
GOTO   END

(Arrête le programme)

HIGH

Cette instruction fait deux choses en même temps :
- après avoir mis en sortie une ligne du Port B (RB), la place aussitôt à l'état logique haut (1).

SYNTAXE :
HIGH *Pin*

↓

Numéro de la pin (0..... 7) du Port B (RB) qu'on veut mettre à l'état haut

EXEMPLE :
HIGH 0
Sort un 1 sur RB0

NB : Seul le numéro de la pin du RB doit être spécifié (exemple : HIGH 7 et non pas HIGH Pin 7), car une instruction ainsi rédigée n'est pas admise, et conduirait à une exécution erronée du programme.

I2Cin

Reçoit des données à partir d'un bus I2C.

SYNTAXE :
I2Cin *Contrôle*, *Adresse*, *Variable*

Variable : Nom de la variable (B0... B51) dans laquelle on veut stocker le résultat de la lecture.

Adresse : Adresse mémoire dont on veut lire le contenu.

Contrôle : Octet composé de :
- la clé d'accès spécifique au circuit I2C utilisé, telle qu'elle a été définie par son fabricant (sur 4 bits)
- 1 bit (celui de plus fort poids) désignant la largeur du bus d'adresses (0 = 8 bits ; 1 = 16 bits)
- 3 bits (ceux de plus faible poids) servant à la sélection des blocs internes (pages de 256 octets).

Cette instruction fait plusieurs choses de suite :
- sort la clé permettant l'accès au circuit I2C
- va à l'adresse voulue
- lit son contenu
- le stocke dans la variable figurant en argument.

EXEMPLE :
I2Cin %01010000, 12, B0
Lit le contenu de l'adresse 12 et le stocke dans la variable B0.

L'octet *Contrôle* (%01010000 ou $50) se lit comme ceci :
- le bit de plus fort poids (0) spécifie un bus d'adresse de 8 bits
- les quatre bits suivants (1010) correspondent à la clé d'accès définie par le fabricant
- les trois bits de plus faible poids, non utilisés dans ce type de mémoire, sont mis à zéro (000).

Cette instruction permet notamment d'ajouter au PIC une mémoire EEPROM série au standard I2C (24LC01B, 24LC02B, 24LC04B, 24LC08B, 24LC16B, 24LC32B, 24LC65... toutes les 24 C0x en général) et de disposer ainsi d'une plus grande taille mémoire.

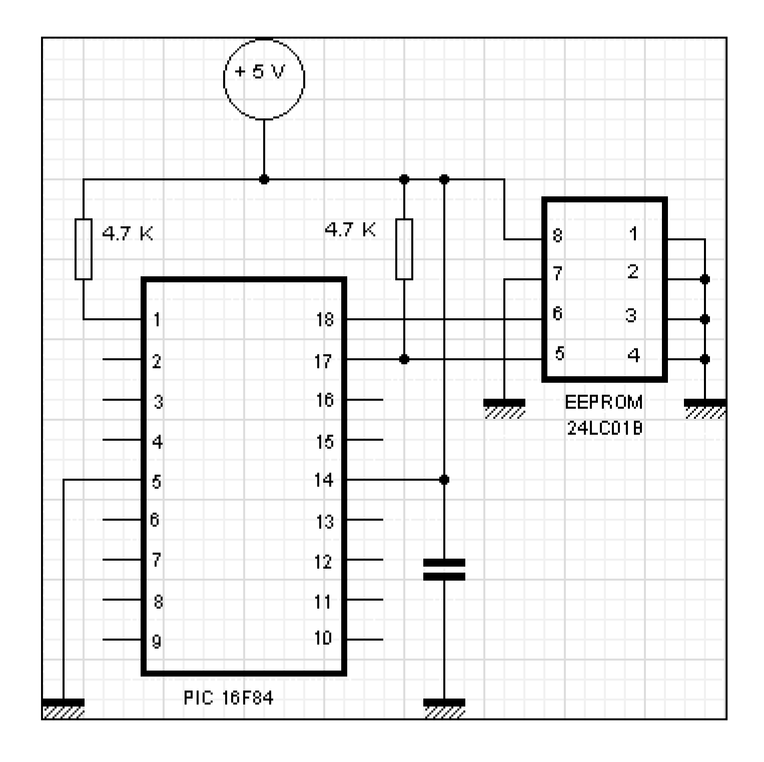
Bien entendu, ce n'est pas seulement à des EEPROM que le PIC peut être associé, mais à n'importe quel circuit du standard I2C (capteur de température, convertisseur A/D, etc...).

Ce tableau donne les clés d'accès des EEPROM les plus courantes :

Modèle	Taille mémoire	*Contrôle*	Largeur de bus
24LC01B	128 octets	%01010xxx	8 bits
24LC02B	256 octets	%01010xxx	8 bits
24LC04B	512 octets	%01010xxb	8bits
24LC08B	1k octets	%01010xbb	8bits
24LC16B	2K octets	%01010bbb	8bits
24LC32B	4K octets	%11010ddd	16 bits
24LC65	8K octets	%11010ddd	16 bits

bbb = bits de sélection du ou des blocs internes (pages)
ddd = bits servant à sélectionner le boîtier
xxx = sans importance

Schéma-type de connexion d'un PIC 16F84
avec une EEPROM 24LC01B



Les lignes *Data* et *Clock* sont affectées respectivement au Port A0 et au Port A1. Ceci dans le but évident de laisser entièrement disponibles les 8 bits du Port B.

La ligne *Data* est bidirectionnelle.
Elle doit comporter une résistance de Pull-up de 4,7 KΩ.

I2Cout

Envoie des données sur un bus I2C.

SYNTAXE :
I2Cout *Contrôle, Adresse,* (*Valeur*)

Valeur à écrire (00... FF)

Adresse à laquelle on veut écrire.

Octet de contrôle composé de :
- la clé d'accès spécifique au circuit I2C, telle q'elle a été définie par le fabricant (sur 4 bits) ;
- 1 bit (celui de plus fort poids) désignant la valeur du bus des adresses (0=8 bits ; 1=16 bits) ;
- 3 bits (ceux de plus faible poids) servant à la sélection des blocs.

L'écriture d'un octet dans une EEPROM série prend environ 10 ms.
Par conséquent, si on veut écrire plusieurs octets en suivant, il faut impérativement respecter ce délai sinon, si une opération d'écriture est encore en route, l'accès à la mémoire est ignoré.
Différent est le cas si, au lieu d'avoir affaire à des mémoires EEPROM, on a affaire à des composants de la famille I2C ne demandant pas un aussi long délai entre deux écritures.

EXEMPLE :
```
I2Cout  %01010000, 17, (42)
PAUSE  10
I2Cout  %0101000, 125 (B3)
PAUSE  10
```

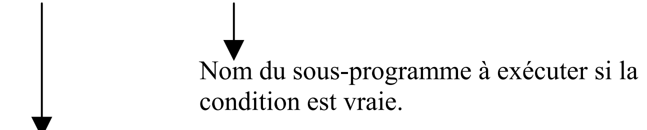
On écrit 42 à l'adresse 17. On laisse passer 10 ms pour que l'opération d'écriture s'achève. Puis on écrit le contenu de la variable B3 à l'adresse 125 et on laisse encore passer 10 ms.
Les lignes *DATA* et *CLOCK* sont affectées respectivement à *PORTA0* et à *PORTA1*. Ceci dans le soucis de laisser entièrement disponibles les 8 bits Port B. La ligne *DATA* est bidirectionnelle. Elle doit comporter une résistance de Pull-up de 4,7 KΩ.

If... Then

Cette instruction teste une condition.
Si la condition est vraie, le programme saute au sous-programme indiqué après *Then*.

Then est ici une sorte de Goto. C'est pourquoi après *Then* on doit impérativement indiquer le nom du sous-programme auquel le programme doit se rendre.
Si la condition est fausse, le programme continue en séquence et analyse l'instruction suivante.

SYNTAXE :
IF *Comparaison* THEN *Label*



Ce que l'on veut comparer :

- ceci égal à cela (=)
- ceci inférieur à cela (<)
- ceci supérieur à cela (>)
- ceci supérieur ou égal à cela (>=)
- ceci différent de cela (<>)

EXEMPLE :
IF Pin0 = 0   THEN OUVERT
IF B0 >=9   THEN PORTE

Si l'interrupteur connecté à la pin 0 du Port A est ouvert (0), exécute le sous-programme appelé OUVERT.
Si la variable B a une valeur supérieure ou égale à 9, exécute le sous-programme appelé PORTE.

INPUT

Place en entrée l'une des 8 lignes du Port B (RB).

SYNTAXE :
INPUT *Pin*

Numéro de la pin du Port B (RB) qu'on veut placer en entrée.

EXEMPLE :
INPUT 3

Met RB3 en entrée.

NB : Seul le numéro de la pin doit être noté. C'est à dire : 0.... 7
et non Pin0.... Pin7

LET

Avec cette instruction on affecte une valeur à une variable.

EXEMPLE :
LET B0 = 37

Affecte à la variable B0 la valeur 37.

LOOKdown

Cherche une valeur (*Recherche*) dans une liste (*Constante, Constante, Constante...*), et fournit en *Index* sa position dans la liste.

SYNTAXE :
LOOKDOWN *Recherche,* (*Constante, Constante, Constante...*), *Index*

L'instruction compare une variable à une liste de constantes jusqu'à ce qu'elle trouve une égalité et – dans ce cas – place dans une variable le numéro du rang de cette valeur.
Si la valeur recherchée est la première de la liste, *Index* prend la valeur 0 ; si c'est la deuxième, *Index* prend la valeur 1, etc..
Si par contre la recherche est infructueuse, aucune action n'a lieu et *Index* reste inchangé.

Par exemple : si la liste des *Constantes* est (3, 12, 14, 27, 9) et la valeur de *Recherche* est 14, *Index* prendra la valeur 2 puisque 14 est la troisième constante de la liste (le décompte commence à 0).

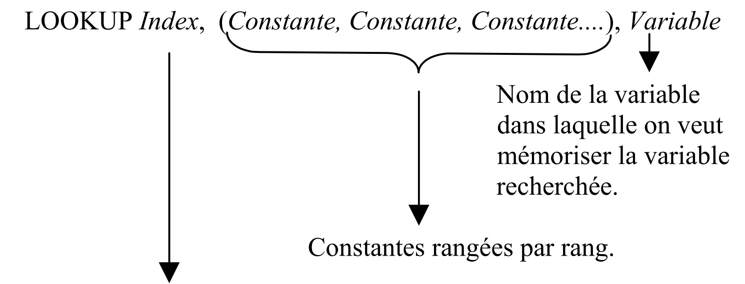
EXEMPLE :
SERin 1, N2400, B0
LOOKDOWN B0, (″ 0123456789ABCDEF″), B1
SERout 0, N2400, (#B1)

Lit les caractères arrivant dans une réception série sur RB1 (pin 1 du Port B), les stocke un à la fois dans la variable B0, et fournit la position de chacun d'eux dans la variable B1, mise ensuite en sortie sur RB0 (pin 0 du Port B).

LOOKup

Cette instruction permet de récupérer une valeur à partir d'une table de constantes.
On donne une valeur à *Index* et on demande à l'instruction de sauvegarder dans une *Variable* la valeur correspondant à son rang dans la liste des variables.
Si *Index* vaut 0, *Variable* assume la valeur de la première constante. Si *Index* vaut 1, *Variable* assume la valeur de la deuxième constante, et ainsi de suite.
Si *Index* est un nombre supérieur au nombre des constantes, il ne se passe rien et *Variable* reste inchangée.

SYNTAXE :
LOOKUP *Index*, (*Constante, Constante, Constante....*), *Variable*



EXEMPLE :

```
FOR B0 = 0 to 6
        LOOKUP B0, ("BONJOUR"), B1
        SERout 0, N2400, (B1)
NEXT B0
```

On donne successivement à B0 les valeurs de 0 à 7 pour récupérer dans la variable B1, l'une après l'autre, toutes les lettres de BONJOUR, lesquelles sortent ensuite en série sur RB1.

LOW

Cette instruction met à l'état logique bas (0) une ligne (0... 7) du Port B (RB).

SYNTAXE :
LOW *Pin*



Numéro de la pin (0... 7) du Port B qu'on veut mettre à l'état bas.

EXEMPLE :
LOW 3

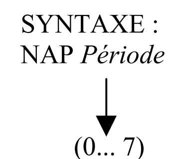
Met à l'état logique bas RB3.

NB : Seul le numéro de la pin de RB doit être spécifié. Exemple : *4*, et non pas *Pin 4*.
Une instruction mal rédigée conduirait à une exécution erronée du programme.

NAP

Cette instruction met le microcontrôleur en veille pendant le temps défini par *Période*.

SYNTAXE :
NAP *Période*



0	18 ms
1	36 ms
2	72 ms
30	144 ms
4	288 ms
5	576 ms
6	1.152 ms
7	2.304 ms

Ces durées sont approximatives car dérivées du Watchdog Timer, piloté par son horloge R/C interne, et donc pouvant différer d'un microcontrôleur à l'autre...

OUTPUT

Cette instruction place en sortie l'une des 8 pins du Port B (RB).

SYNTAXE :
OUTPUT *Pin*

↓

Numéro de la pin du Port B (RB) qu'on veut placer en sortie.

EXEMPLE :
OUTPUT 3

Met RB3 en sortie.

NB : Seul le numéro de la pin de RB doit être spécifié. Exemple : 3, et non pas *Pin 3*.
Une instruction mal rédigée conduirait à une exécution erronée du programme.

PAUSE

Instruction introduisant une temporisation.

SYNTAXE :
PAUSE *Période*

↓

Durée, exprimée en millisecondes.

Pour un 16F84 cadencé à 4 MHz, l'unité de temps est approximativement 1 milliseconde.

EXEMPLE :
PAUSE 500

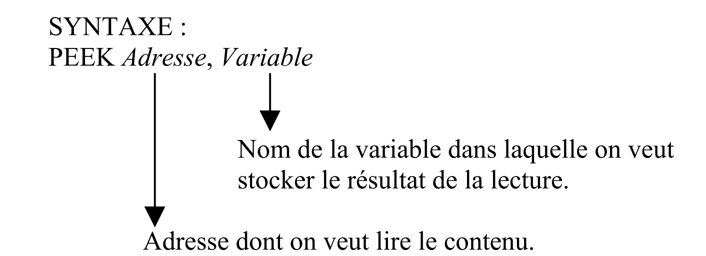
Engendre un temporisation de 500 ms (½ seconde).

Contrairement à l'instruction NAP qui fournit des délais prédéfinis et approximatifs (car issus du Watchdog Timer), l'instruction PAUSE fournit des délais à la demande et plus précis (bien qu'encore pas tout à fait exacts, sa précision dépendant entre autre de celle du quartz et de l'état de l'alimentation).

PEEK

Cette instruction lit le contenu d'une adresse et le stocke dans une variable.

SYNTAXE :
PEEK *Adresse*, *Variable*



EXEMPLE :
PEEK PORTA, B0

On lit le contenu du Port A et on le mémorise dans la variable B0.

Autre EXEMPLE :

```
Symbol  PORTA = 5
Symbol  TRISA = $85
        POKE  TRISA, 255
        PEEK    PORTA, B0
```

Ensuite, on peut continuer comme ceci :

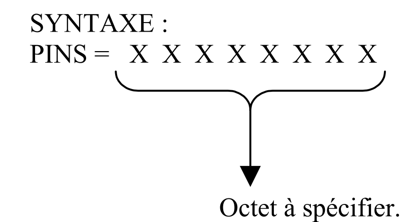
```
IF BIT0 = 0 THEN......
IF BIT0 = 1 THEN......
```

On lit le Port A par l'intermédiaire de la variable B0. Ensuite on peut tester et agir en fonction de l'état de chaque bit.

PINS

Cette instruction met en sortie, sur le Port B, l'octet spécifié après le signe =

SYNTAXE :
PINS = X X X X X X X X



Octet à spécifier.

EXEMPLES :

1) PINS = $2B
Met la valeur hexadécimale 2B (00101100) sur le Port B.
Ce qui a pour effet de mettre :
le bit 0 à 0
le bit 1 à 0
le bit 2 à 1
le bit 3 à 1
le bit 4 à 0
le bit 5 à 1
le bit 6 à 0
le bit 7 à 0.

2) PINS = B1
Met chacun des bits du Port B à l'état haut ou à l'état bas, conformément à l'octet se trouvant dans la variable B1.

3) PINS = 255
Met à l'état haut tous les bits du Port B.

4) DIRS = 255 (Port B en sortie)

```
LOOP    FOR B1 = 0 TO 255
        PINS = B1
        PAUSE 500
        NEXT B1
        GOTO LOOP

        END
```

On affiche sur le Port B toutes les valeurs de 0 à 255
(FOR B1 = 0 TO 255... NEXT B1) à un intervalle de 500 ms.
Puis on recommence indéfiniment.

POKE

C'est l'instruction par laquelle on écrit une donnée à l'adresse que l'on désigne.

SYNTAXE :
POKE *Adresse*, *Donnée*

Donnée : Donnée à écrire.

Adresse : Adresse à laquelle on veut écrire la donnée.

EXEMPLES :

1) POKE TRISA, 0
Met 0 dans TRISA.

2)

```
SYMBOL    PORTA = 5        (Adresse de PORTA)
SYMBOL    TRISA = $85      (Adresse de TRISA)

          POKE TRISA, 0     (Port A en sortie)
          PEEK PORTA, B0    (Copie Port A
                             dans B0)
          BIT1 = 1
          BIT2 = 0
          etc...
          POKE PORTA, B0

          END
```

Ainsi faisant, on copie sur le Port A ce qui se trouve dans la variable B0.

POT

Lit la valeur d'une résistance (potentiomètre, thermistance, jauge de contrainte, capteur de position ou autre composant résistif) sur l'une des pins (0..... 7) du Port B.

La valeur de la résistance (pouvant aller de 5 KΩ à 50 KΩ) est déterminée, en fait, en mesurant le temps de décharge du condensateur associé à la résistance avec laquelle il forme une cellule R/C.

SYNTAXE :
POT *Pin*, *Echelle*, *Variable*

Variable : Nom de la variable dans laquelle on veut mémoriser le résultat de la mesure. Si *Echelle* est positionné à sa valeur optimale, *Variable* prend une valeur proche de 0 lorsque le potentiomètre est tourné sur le minimum, et une valeur proche de 255 lorsque le potentiomètre est tourné sur le maximum.

Echelle : (1..... 255) est un facteur de conversion qui dépend des caractéristiques de la cellule R/C. Sa valeur doit être déterminée expérimentalement. Pour de grandes constantes de temps R/C, la valeur d'*Echelle* doit être petite (le minimum est 1). Au contraire, pour de petites constantes de temps, la valeur à donner à *Echelle* doit être grande (255).

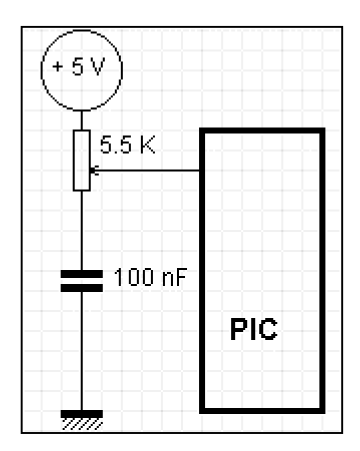
Pin : (0....7) Désigne la pin du Port B (RB) sur laquelle on veut lire la valeur de la résistance.

Pour déterminer au plus juste la valeur qu'il convient de donner à *Echelle*, la méthode consiste à la faire évaluer par le PIC lui-même. Pour cela, réglez d'abord la valeur de la résistance à son maximum, et lisez-la avec *Echelle* provisoirement positionnée à 255. Cette valeur (stockée dans *Variable*) évaluée par le PIC, est celle que vous pourriez alors donner à *Echelle* (valeur optimale).
Evidemment, pas question de s'attendre à lire une valeur précise de Résistance, exprimée en Ohms !

Ici, en définitive, on lit des paliers. Mais la valeur qu'assume tour à tour *Variable* chaque fois qu'on tourne l'axe du potentiomètre, renseigne suffisamment sur la position de l'axe et – par là même – sur la valeur prise par la résistance, si on connaît sa valeur de butée.

EXEMPLE :
POT 7, 255, B0
SERout 0, N2400, (#B0)

Lit un potentiomètre relié à RB7, mémorise le résultat dans la variable B0, puis met cette valeur en sortie sur RB0, en mode série.
Le schéma de branchement est le suivant :



PULSin

Mesure la durée d'une impulsion arrivant sur une pin du Port B (RB).
La pin désignée est automatiquement mise en entrée.
NB : La durée est exprimée par unités de 10 µs. Autrement dit : le résultat de la mesure est à multiplier par 10 pour avoir le nombre de microsecondes recherché.

SYNTAXE :
PULSin *Pin, Etat, Variable*

Variable : Nom de la variable à 16 bits (donc : de type W) dans laquelle on veut mémoriser le résultat de la mesure.
Cette variable peut aller de 1 à 65.535 (correspondant respectivement à :
- 1 x 10 µs = 10 µs
.
.
- 65.535 x 10 µs = 655.350 µs (soit 655,35 ms ou encore 0,65535 secondes).

Etat : (0 ou 1)
Définit la partie du signal dont on veut lire la durée :

1 = la partie haute
0 = la partie basse.

Pin : (0... 7)
Numéro de la pin du Port B (RB) qui reçoit le signal dont on veut mesurer la durée.
Attention : seul le numéro de la pin doit être noté (0..... 7).

EXEMPLE :
PULSin 5, 1, W0
Mesure la durée de la partie haute d'une impulsion arrivant sur RB5 et stocke le résultat dans la variable (à16 bits) W0.

NB : Si aucun front n'est détecté, ou si l'impulsion dure plus longtemps que 0.65535 seconde, *Variable* est mise à 0.

Si on définit une variable de type B (variable à 8 bits) au lieu d'une variable de type W (variable à 16 bits), seuls les 8 bits de poids faible sont mémorisés.

PULSout

Génère - sur une pin du Port B - une impulsion de durée calibrée, exprimée par unités de 10 µs.
La pin désignée est automatiquement mise en sortie.
Le niveau (haut ou bas) de l'impulsion calibrée mise en sortie, dépend de l'état de la pin avant l'exécution de cette instruction, car - lorsque cette instruction survient - elle inverse l'état actuel, pour marquer un début à partir duquel elle peut calculer la durée.

SYNTAXE :
PULSout *Pin*, *Période*

Durée de l'impulsion (de 1 à 65.535) par pas de 10 µs.
1 correspond à (1x10 µs) soit 10 µs
2 correspond à (2x10 µs) soit 20 µs
.
65.535 correspond à (65.535x10 µs) soit 0.65535 secondes.

(0... 7) Désigne le numéro de la pin du Port B (RB) sur laquelle on veut sortir l'impulsion calibrée.
Attention : seul le numéro (1, 2, 3..... 7) de la pin doit être noté, et non pas Pin 1, Pin 2...... Pin 7

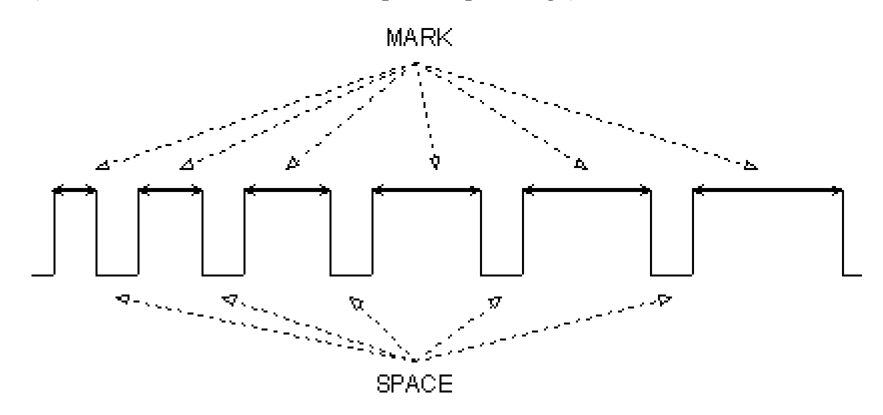
EXEMPLE :
PULSout 4, 100
Envoie sur RB4 une impulsion de 1 ms (100x10 µs).

PWM

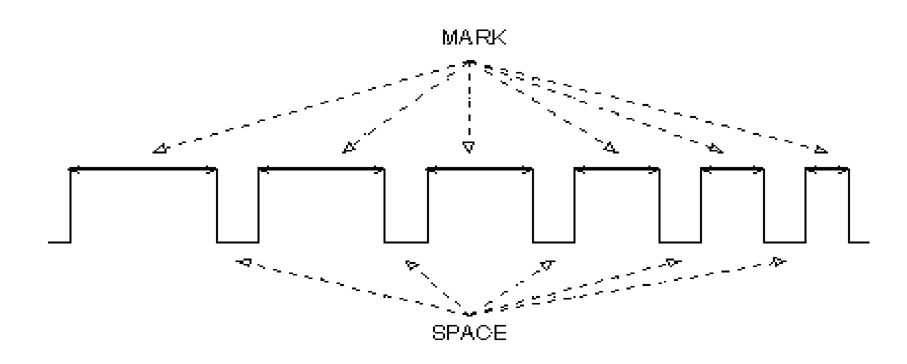
Pulse With Modulation

Modulation par largeur d’impulsions

Exemple de largeurs d’impulsions allant en augmentant
(ici l’état haut *MARK* devient de plus en plus large) :



Exemple de largeurs d’impulsions allant en diminuant
(ici l’état haut *MARK* devient de plus en plus court) :



Cette instruction génère sur une pin du Port B (RB) un train d’impulsions modulées en largeur.

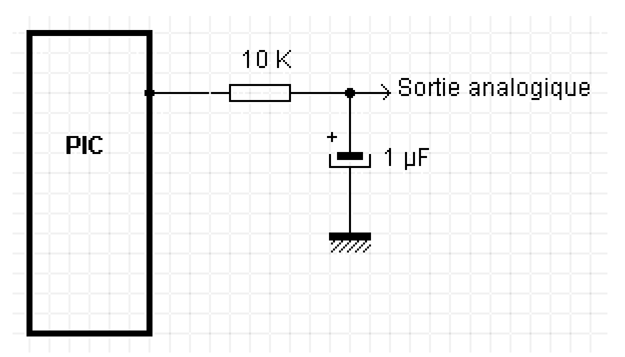
La pin désignée est automatiquement mise en sortie mais, sitôt le cycle terminé, elle est automatiquement remise en entrée.

SYNTAXE :
PWM *Pin*, *Duty*, *Cycle*

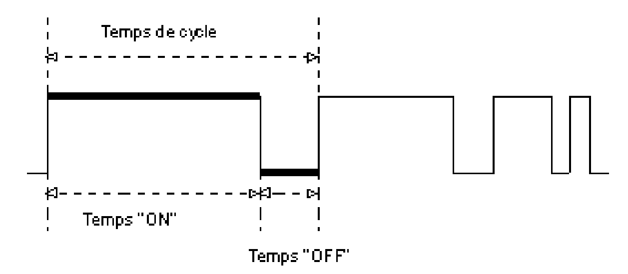
- *Cycle* : Nombre d'impulsions qu'on veut élargir ou retrécir.
- *Duty* : Désigne (en pourcentage) le cycle de travail de chaque train d'impulsions (peut varier de 0% à 255%). C'est le coefficient d'élargissement ou de retrécissement qu'on veut appliquer aux impulsions.
- *Pin* : Numéro de la pin du Port B (RB) sur laquelle on veut générer le train d'impulsions PWM.

Une LED commandée par un signal PWM voit sa luminosité s'accroître progressivement ou diminuer progressivement (selon le mode de programmation), comme si elle était associée à un variateur, produisant ainsi une sorte d'effet crépucsulaire.

Moyennant un réseau R/C, cette instruction permet de mettre en place un convertisseur D/A tout simple :



La modulation par largeur d’impulsions (modulation d’impulsions en durée, ou modulation par nombre d’impulsions dans le temps) est une technique qui consiste à faire varier la puissance moyenne de sortie, dans le temps, en agissant sur le rapport cyclique (temps ON / temps de cycle).
Le temps de cycle est un état ON plus un état OFF.



La puissance moyenne du signal diminue (ou augmente) en fonction du temps.
Le temps de cycle est constant, mais le temps « ON » varie (sa durée diminue, ou augmente).

Ce type de modulation présente au moins trois avantages : un seul bit de Port suffit à commander les transitions ON/OFF ; le signal ainsi modulé réduit la dissipation de puissance (perte de chaleur, par exemple), et peut commander une charge à puissance variable.

Temps de cycle = un temps « ON » + un temps « OFF ».

$$\text{Rapport cyclique} = \frac{\text{Temps « ON »}}{\text{Temps de cycle}}$$

Un exercice pour vous entraîner à utiliser cette instruction consiste à brancher un voltmètre pour visualiser et mesurer un signal PWM, en y associant une LED servant (dans une certaine mesure) de confirmation.
Le voltmètre sera un modèle analogique (à aiguille) de préférence à un modèle digital.
La tension de sortie moyenne est proportionnelle au rapport cyclique et indépendante du temps de cycle.
Avec un temps de cycle maximum, vous risquez de voir l’aiguille du voltmètre se déplacer très légèrement.

Un autre exercice pourrait consister à faire varier le temps « ON » et le temps « OFF » en fonction des données d'un tableau.
Avec ces prescriptions, par exemple :

Rapport cyclique	Temps « ON »	Temps « OFF »	Temps de cycle
0,10	20	180	200
0,20	20	80	100
0,30	30	70	100
0,40	40	60	100
0,50	60	60	120
0,60	90	60	150
0,70	140	60	200
0,80	160	40	200
0,90	180	20	200
1,00	200	0	200

RANDOM

Génère un nombre aléatoire sur 16 bits et le stocke dans une variable de type W (variable de 16 bits, composée de la juxtapposition de deux variables de 8 bits. Exemple : la variable W0 est composée de la juxtapposition des variables B0 et B1).

SYNTAXE :
RANDOM *Variable*

Nom d'un variable de 16 bits (W0.... W25) dans laquelle le programme va stocker le nombre aléatoire.

EXEMPLE :
RANDOM W0
Génère un nombre aléatoire de 16 bits et le stocke dans la wariable W0.

NB : Le nombre aléatoire pouvant être généré par cette instruction peut aller de 1 à 65.535.
Le nombre 0 n'est pas généré.

READ

Avec cette instruction on lit une donnée dans la mémoire EEPROM.
Le résutlat de la lecture est mémorisé dans la variable spécifiée.

SYNTAXE :
READ *Adresse*, *Variable*

Variable : Nom de la variable (B0... B51) dans laquelle on veut stocker le résultat de la lecture.

Adresse : Adresse EEPROM dont on veut lire le contenu.

Rappel : Pour écrire dans cette mémoire on utilise l'instruction EEPROM.

RETURN

C’est l’instruction qui termine tout sous programme.
Un sous programme commence par GOSUB, et doit obligatoirement se terminer par RETURN.
Une fois le sous programme terminé, cette instruction provoque le retour au programme principal, à l’instruction se trouvant just’après le GOSUB ayant appelé le sous programme.

REVERSE

Inverse le sens de fonctionnement de la patte spécifiée (0.... 7)
du Port B (RB).
Si elle était une *entrée*, après cette instruction elle devient une *sortie*,
et inversement (si elle était une sortie, elle devient une entrée).

SYNTAXE :
REVERSE *Pin*

↓

Numéro (0.... 7) de la pin du Port B (RB) dont on veut inverser le sens.

EXEMPLE :
OUTPUT 7
REVERSE 7
Met d'abord RB7 en *sortie*, puis inverse le sens
et met RB7 en *entrée*.

NB : Seul le numéro (0.... 7) de la pin du Port B (RB) doit être noté
(REVERSE 0, REVERSE 1,....... REVERSE 7
et non REVERSE Pin0, REVERSE Pin1.....).

SERin

Serial Input

Cette instruction permet de recevoir des données sous forme série asynchrone , par mots de 8 bits, sans parité, avec un seul bit de STOP. La réception doit avoir lieu en utilisant l'une des pins (0.... 7) du Port B (RB).

SYNTAXE :
SERIN *Pin*, *Mode*, (*Qualificateur*), *Variable*

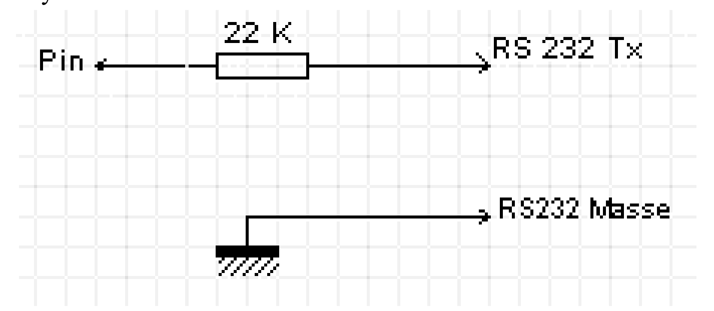
Variable : Nom de la variable (B0... B51) dans laquelle on veut stocker les données reçues.

Qualificateur : A noter entre parenthèses. Ca peut être un seul, ou plusieurs. Si plusieurs, une virgule doit les séparer. Il s'agit des informations optionnelles (formant une sorte de « clé ») qui doivent être reçues exactement dans l'ordre spécifié, avant que la ou les données ne soient prises en compte. Si, lors de la réception de plusieurs *Qualificateurs*, un seul ne correspond pas à la clé à laquelle ils sont comparés un à un, le processus revient au début, et le programme reprend à les examiner depuis le premier. Un *Qualificateur* peut être une constante, une variable ou une chaîne de caractères (dans ce dernier cas, chaque caractère de la chaîne est traité individuellement, pris en sa valeur ASCII). Cela sert à sécuriser une transmission. Très utile notamment dans le cas des télécommandes, d'alarmes ou d'anti-vol à distance).

Mode : Définit à la fois la vitesse de transmission (Baud Rate) et la forme des données (TTL vrai ou TTL inversé). On peut choisir parmi 8 modes possibles, selon les spécifications du tableau suivant :

Symbole	Valeur	Baud Rate	Mode
T2400	0	2400	TTL vrai
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	TTL inversé
N1200	5	1200	
N9600	6	9600	
N300	7	300	

Etant donné que les circuits d'interface RS232 (même les plus économiques) présentent d'excellentes caractéristiques d'entrée/sortie, les signaux peuvent être gérés par les PIC sans besoin d'un convertisseur de niveaux.
Les signaux présentés en format TTL inversé doivent être utilisés moyennant une résistance de limitation de courant :



EXEMPLE :

```
LOOP        SERin 7, n1200, B0
            IF B0>0  THEN SORTIE
SORTIE      POKE Pin3, B0
            PAUSE 300
            GOTO LOOP
```

Reçoit un signal série sur RB7, en TTL inversé, à la vitesse de 1200 Bauds, et le stocke dans la variable B0.
Si B0 contient une donnée, elle est envoyée sur la pin 3 du Port A (RA3). On temporise 300 ms. Puis on recommence avec l'octet suivant.

SERout

Serial Output

Cette instruction permet d'envoyer des données, sous forme série asynchrone, par mots de 8 bits, sans parité, avec un seul bit de STOP.
L'émission doit avoir lieu en utilisant l'une des pins (0... 7) du Port B (RB).

SYNTAXE :
SERout *Pin*, *Mode*, (*Donnée, Donnée....*)

A noter entre parenthèses.
Ca peut être une ou plusieurs.
Si plusieurs, une virgule doit les séparer.
Il s'agit des informations à transmettre.
Ces informations peuvent être des constantes, des variables ou une chaîne de caractères.

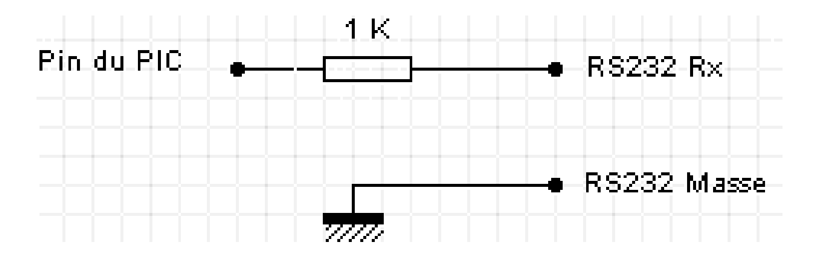
- Une chaîne de caractères est traitée comme une suite de caractères. Chacun d'eux est émis individuellement.
- Une valeur numérique (variable ou constante) est émise sous la forme de son équivalent ASCII. Ainsi, par exemple, 13 est un retour chariot et 10 est un saut de ligne.
- Une valeur numérique précédée du signe # (dièse) est émise sous la forme de la représentation en ASCII de la valeur décimale correspondante. Ainsi par exemple, #123 fera émettre « 1 », puis « 2 », puis « 3 ».

Définit à la fois la vitesse de la transmission (Baud Rate), la forme des données (TTL vrai ou TTL TTL inversé) et la configuration de l'étage de sortie (Drain ouvert ou Collecteur ouvert).
On peut choisir parmi 16 modes possibles, selon les spécifications du tableau suivant :

Symbole	Valeur	Baud Rate	Mode
T2400	0	2400	TTL vrai
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	TTL inversé
N1200	5	1200	
N9600	6	9600	
N300	7	300	
OT2400	8	2400	Drain ouvert
OT1200	9	1200	
OT9600	10	9600	
OT300	11	300	
ON2400	1	2400	Source ouvert
ON1200	2	1200	
ON9600	13	9600	
ON300	14	300	

Pin désigne le numéro (0... 7) de la pin du Port B (RB) que l'on veut utiliser pour sortir le signal série.

Etant donné que les circuits d'interface RS232 (même les plus économiques) présentent d'excellentes caractéristiques d'entrée/sortie, les signaux peuvent être gérés par les PIC sans besoin d'un convertisseur de niveux.
Les signaux présentés en format TTL inversé doivent être utilisés moyennant une résistance de limitation de courant :



	Sur fiche DB9	Sur fiche BD25
RS232 Rx	Pin 2	Pin 3
RS233 Masse	Pin 5	Pin 7

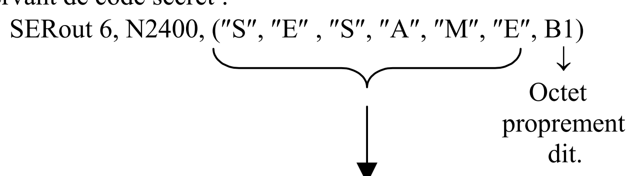
EXEMPLES :

1) Un seul octet à transmettre :
SERout 6, N2400, (B1)

2) Plusieurs octets à transmettre :
SERout 6, N2400, (B1, B0.....0F....C3)

3) Envoyer sur RB0 la valeur ASCII contenue dans la variable B0, suivie d'un saut de ligne :
SERout 0, N2400, (#B0, 10)

4) Envoi d'un octet (stocké dans la variable B1) précédé d'une clé servant de code secret :
SERout 6, N2400, (″S″, ″E″ , ″S″, ″A″, ″M″, ″E″, B1)



Caractères servant de clé (caractères « secrets »).

Cette façon de coder est particulièrement utile dans le cas de systèmes d'antivol.
Le récepteur, sans avoir au préalable reçu le mot SESAME, ne peut mémoriser le mot fourni par B1.

SLEEP

Met le PIC en mode veille pendant un certain temps (défini par *Période*) exprimé en secondes.
Période est une variable de 16 bits, pouvant aller de 1 à 65.535 (plus de 18 heures) avec une précision qui dépend du timer interne (celui associé au chien de garde) qui est du type R/C. Cette précision n'est qu'approximative, et ne doit pas être utilisée comme référence temporelle absolue.
En plus, il faut savoir que la plus courte *Période* ne peut être inférieure à celle du time-out maximum du chien de garde, qui est de 2,3 secondes.
En mode SLEEP, le microcontrôleur passe en mode basse consommation (low power).
Lorsque le délai spécifié est écoulé, l'exécution du programme reprend avec l'instruction suivante.

SYNTAXE :
SLEEP *Période*



Durée de veille, éxprimée en secondes.

EXEMPLE :
SLEEP 60
Met le µC en veille pendant 1 minute (60 secondes).

SOUND

Génère un son sur la pin désignée du Port B (RB).

SYNTAXE :
SOUND *Pin*, (*Note*, *Durée*, *Note*, *Durée*,...)

Durée de la note (ms).

Hauteur de la note: - de 1 à 127 = notes audibles
- de 128 à 256 = bruit blanc.

Numéro (0.... 7) de la pin du Port B (RB) sur laquelle on veut faire sortir le son.

EXEMPLES :

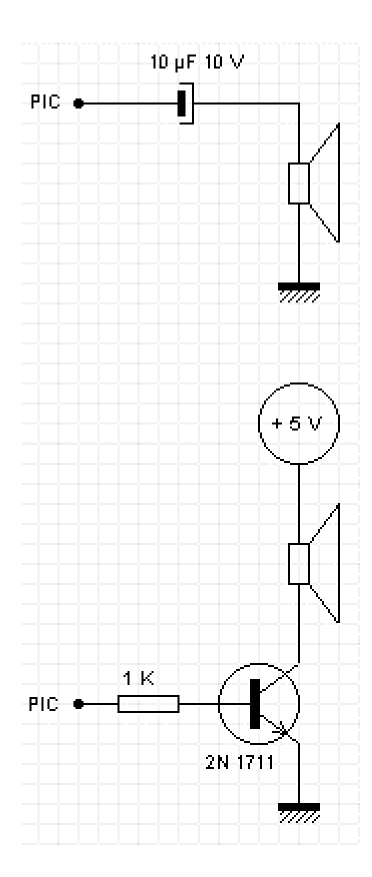
1) SOUND 7, (100, 50)
Génère sur RB7 une note de hauteur 100, pendant 50 ms.

2) SOUND 7, (1, 100, 65, 100, 127, 100)
Génère sur RB7 trois notes : une note grave pendant 100 ms, suivie d'une note moyenne pendant 100 ms, suivie encore d'une note aigüe pendant 100 ms.

Le son généré a la forme d'un signal carré.
La *Durée* et la hauteur de la *Note* sont à déterminer expérimentalement, car elles peuvent varier d'un µC à l'autre en fonction notamment de la fréquence du quartz pilote, mais aussi de la tension d'alimentation, de la température, etc...

Les paramètres enfermés dans les parenthèses doivent indiquer la note et sa durée, la note et sa durée, la note et sa durée....
Leur nombre doit être (forcément !) pair.
On peut mettre dans les parenthèses autant de paramètres qu'on veut.
Il y a – bien sûr – une limite : la taille mémoire maximale du PIC !

Voici deux façons possibles de raccorder un haut-parleur à une pin du Port B :



TOGLE

Cette instruction inverse l'état logique de l'un des bits (0... 7)
du Port B (RB).

SYNTAXE :
TOGLE *Pin*
↓
Numéro de la pin (0.... 7) du Port B dont on veut inverser l'état.

EXEMPLE :
LOW 0
TOGLE 0
Dans un premier temps on met RB0 à l'état bas,
puis on inverse l'état de RB0 (RB0 passe
de l'état bas à l'état haut).

TRISA

Configure le Port A (en *sortie* ou en *entrée*).

Exemples :

```
1) SYMBOL    PORTA = 5
   SYMBOL    TRISA = $85
   POKE      TRISA, 0
```

On configure le Port A en *sortie*

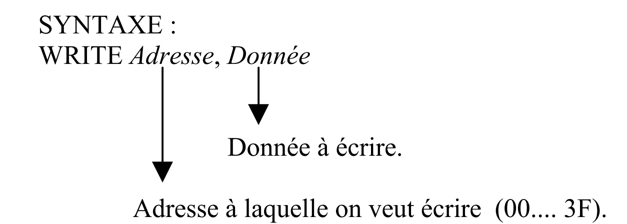
```
2) SYMBOL    PORTA = 5
   SYMBOL    TRISA = $85
   POKE      TRISA, 255
```

On configure le Port A en *entrée*

WRITE

C'est l'instruction avec laquelle on écrit dans la mémoire EEPROM de données (Data Memory).

SYNTAXE :



WRITE *Adresse*, *Donnée*

Donnée à écrire.

Adresse à laquelle on veut écrire (00.... 3F).

EXEMPLE :
WRITE 6, B0
Ecrit à l'adresse 6 la donnée se trouvant dans la variable B0.

NB : Les données entrent dans la mémoire EEPROM une seule fois au moment de la programmation du PIC et non pas chaque fois qu'on exécute le programme.

Exemples de programmes écrits en langage PicBASIC

Voici 17 exemples de programmes écrits en MEL PicBASIC qu'ils ne faut pas juger au nombre d'instructions, car – ainsi que je l'ai déjà dit – les programmes écrits en PicBASIC sont souvent courts, cela étant dû à la puissance de ses instructions.

Ce ne sont que des bases. Ils illustrent le mécanisme de la programmation en MEL PicBASIC (je rappelle que MEL est l'acronime de ***M***ICRO ***E***NGINERING ***L***ABS, la Société qui a développé ce BASIC pour PIC).

Je vous encourage à vous procurer un Editer MEL PicBASIC
et à faire comme moi.
Vous verrez que développer avec ce logiciel devient un jeu d'enfants.

Bonne programmation.

Programme 1

Faire clignoter une LED reliée à RB0 (pin 0 du Port B).

```
LOOP :  HIGH 0
        PAUSE 1000
        LOW 0
        PAUSE 1000
        GOTO   LOOP
        END
```

La LED s'allume pendant 1 seconde (1000 ms), puis s'éteint pendant 1 seconde.
Puis le cycle recommence indéfiniment.

Pour que la LED clignote plus vite, il faut ajuster la durée de PAUSE.
Exemple :

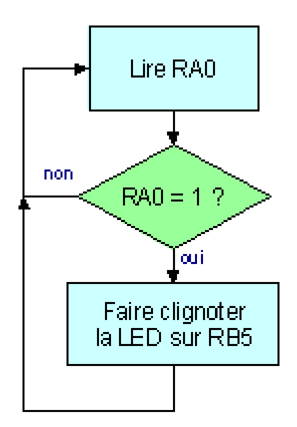
```
LOOP :  HIGH 0
        PAUSE 100
        LOW 0
        PAUSE 100
        GOTO   LOOP
        END
```

Dans ce cas la LED s'allume pendant 100 ms, puis s'éteint pendant 100 ms. Puis le cycle recommence indéfiniment.

Programme 2

Faire clignoter une LED relée à RB5 (bit 5 du Port B) **si l'interrupteur relié à RA0** (bit 0 du Port A) **est à l'état haut.**

```
LOOP :  IF PIN 0 = 1 THEN LED
        GOTO LOOP
LED :   HIGH 5
        PAUSE 100
        LOW 5
        PAUSE 100
        GOTO LOOP
        END
```



PAUSE étant ici positionnée à 100, la LED clignote plutôt rapidement.

Pour faire clignoter la LED moins rapidement, il suffit de modifier la valeur de PAUSE :

```
LOOP :  IF PIN 0 = 1 THEN LED
        GOTO LOOP
LED :   HIGH 5
        PAUSE 1000
        LOW 5
        PAUSE 1000
        GOTO LOOP
        END
```

1000 ms = 1 seconde. Ici la LED clignote lentement.

On y remarque deux labels : LOOP et LED. Chacune renvoit à un sous programme.
Les noms des labels sont suivis du signe : (deux points) :
- LOOP :
- LED :

Dans le premier sous programme on dit :

Si la pin 0 du Port A est à l'état 1, cesse d'exécuter les instructions en ligne et saute au sous programme LED, sinon (ligne suivante GOTO LOOP) surveille l'état logique de cette pin. Si elle est à l'état 0, continue à surveiller.

Dans le deuxième sous programme on dit :

Allume la pin 0 du Port B pendant 1 seconde, puis éteins-la pendant 1 seconde. Puis recommence la scrutation de l'état de la Pin 0 du Port A.

Les pins auxquelles le langage MEL PicBASIC fait tacitement référence sont celle du Port B.
HIGH 5 signifie : met à l'état haut la pin 5 du Port B.
HIGH (mets à l'état haut)
5 (la pin 5 du Port B).

Programme 3

Faire clignoter une LED reliée à RB5 (pin 5 du Port B) **si les interrupteurs reliés à RA0 et à RA1 sont fermés.**

```
LOOP :  IF Pin 0 = 1 AND Pin 1 = 1 THEN LED
        GOTO LOOP
LED :   HIGH 5
        PAUSE 100
        LOW 5
        PAUSE 100
        GOTO LOOP
        END
```

Si la pin 0 (RA0) et la pin 1 (RA1) du Port A sont à 1, on exécute le sous programme LED, sinon on continue à surveiller l'état de RA0 et de RA1.

Lorsque les conditions sont remplies, on allume la LED reliée à RB5 (pin 5 du Port B) pendant 100 ms, puis on l'éteint pendant 100 ms. Puis on revient au début pour voir si les deux interrupteurs sont fermés. Si Pin0 et Pin1 ne sont pas tous les deux enfoncés, il ne se passe rien (la LED reste éteinte).

Programme 4

Allumer successivement toutes les LED du Port B.

```
LOOP :  FOR B0 = 0 TO 7
        HIGH B0
        NEXT B0
        END
```

Programme 5

Allume et éteint en séquence, une à la fois, toutes les LED du Port B (de RB0 à RB7), **avec un petit intervalle de temps entre chaque.**

```
LOOP :  FOR B0 = 0 TO 7
        HIGH B0
        PAUSE 100
        LOW B0
        PAUSE 900
        NEXT B0
        GOTO LOOP
        END
```

Programme 6

Faire clignoter simultanément toutes les 8 LED du Port B.

On pourrait certes écrire :

```
LOOP :  HIGH 0
        HIGH 1
        HIGH 2
        HIGH 3
        HIGH 4
        HIGH 5
        HIGH 6
        HIGH 7
        PAUSE 100
        LOW 0
        LOW 1
        LOW 2
        LOW 3
        LOW 4
        LOW 5
        LOW 6
        LOW 7
        PAUSE 100
        GOTO LOOP
        END
```

Pas très beau !

L'utilisation d'une variable simplifie l'écriture :

```
LOOP :  FOR B0 = 0 TO 7
        HIGH B0
        NEXT B0
        PAUSE 100
        FOR B0 = 0 TO 7
        LOW B0
        NEXT B0
        PAUSE 100
        GOTO LOOP
        END
```

Les deux programmes font exactement la même chose, mais le deuxième utilise beaucoup moins d'instructions.

Faire des économies d'instructions dans un si court programme n'a pas de sens ; mais cela vaut la peine quand on écrit des programmes longs. Exercez-vous donc à toujours recourir aux instructions les mieux adaptées.

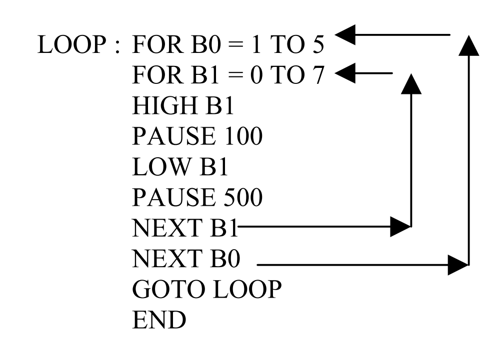
Programme 7

Faire clignoter 5 fois toutes les LED du Port B ensemble.

```
LOOP :  FOR B1 = 0 TO 7
        HIGH B1
        PAUSE 50
        LOW B1
        PAUSE 500
        NEXT B1
        FOR B0 = 1 TO 5
        HIGH B1
        PAUSE 50
        LOW B1
        PAUSE 50
        NEXT B1
        NEXT B0
        GOTO LOOP
```

Programme 8

Faire clignoter en séquence, 5 fois chacune, l'une après l'autre et une à la fois, les 8 LED du Port B, en commençant par la LED associée à RB1.... jusquà celle associée à RB7. Puis recommencer.



Ce programme fait clignoter toutes les LED du Port B (0 à 7) cinq fois, l'une après l'autre, à un intervalle de demi-seconde, créant l'illusion d'une LED qui se décale en clignotant.

Programme 9

Allumer les 8 LED du Port B en séquence binaire.

```
          DIRS = 255
LOOP :    FOR B1 = 0 TO 255
          PINS = B1
          PAUSE 200
          NEXT B1
          GOTO LOOP
```

Dans ce programme :
- DIRS = 255 définit les 8 lignes du Port B comme *sorties* ;
- Le comptage se fait de 0 à 255 (décimal)
  (on pourrait écrire : FOR B1 = %0 TO %11111111
  ou aussi : FOR B1 = $0 TO $FF) ;
- A chaque incrément de la valeur assumée par la variable B1 (de 0 à 255 en décimal) l'instructions PINS = B1 visualise la valeur courante de B1 ;
- L'instruction PAUSE 200 maintient cette visualisation active pendant 200 ms.

Programme 10

Générer sur RB3 un signal modulé en largeur d'impulsions (PWM) sous le contrôle de 2 interrupteurs :

- **l'un placé sur RA0, faisant augmenter la luminosité de la LED lorsqu'il est fermé ;**
- **l'autre placé sur RA1, faisant diminuer la luminosité de la LED lorsqu'il est fermé.**

NB : Un transistor peut être branché sur RB3 pour commander une charge plus importante, ou un triac pour commander une lampe secteur.

```
SYMBOL      PORTA = 5
SYMBOL      TRISA = $85
            POKE TRISA, 255
            B1 = 127
LOOP :      PEEK PORTA, 0
            PWM 3, B1, 10
            IF Bit 0 = 1 THEN INC
            IF Bit 1 = 1 THEN DEC
            GOTO LOOP
INC :       IF B1 > 250 THEN LOOP
            B1 = B1 + 5
            GOTO LOOP
DEC :       IF B1 < 5 THEN LOOP
            B1 = B1 – 5
            GOTO LOOP
            END
```

Les trois premières lignes sont des initialisations . On y dit que PORTA est à l'adresse 5, TRISA est à l'adresse 85, et qu'on veut tous les bits du Port A en entrée.
B1 = 127 donne à la LED une luminosité initiale moyenne (127 est à la moitié entre 1 et 256).
PEEK PORTA, 0 lit l'état de RA0.
Puis on définit les paramètres de la PWM, avec un coefficient changeant de 5% par coup, sur 10 cycles, et si RA0 est à 1 on incrémente la variable B1 (en appelant le sous programme INC), tandis que si RA1 est à 1, on décrémente la variable B1 (en appelant le sous programme DEC).

Programme 11

Réaliser une syrène à 2 tons.

```
LOOP    SOUND 7, (10, 100, 50, 100)
        GOTO LOOP
        END
```

Ce programme génère sur RB7 un son de hauteur 10 pendant 100 ms,
suivi d'un deuxième de hauteur 50 pendant 100 ms.
Puis on recommence.

Programme 12

Générer une mélodie répétitive à 3 notes.

```
LOOP    SOUND 7, (10, 100, 60, 100, 120, 100)
        GOTO LOOP
        END
```

Ce programme génère sur RB7 un premier son de hauteur 10 pendant (environ) 100 ms, puis un deuxième son de hauteur 60 pendant (approximativement) 100 ms, et enfin un troisième son de hauteur 120 pendant encore (approximativement) 100 ms.
Puis on recommence sans cesse.

Programme 13

Générer une mélodie en *crescendo* de 20 notes jouées en continuation.

```
LOOP   B0 = 0
INC    FOR B0 = B0 + 5
       SOUND 7, (B0, 100)
       NEXT B0
       IF B0 > 100 THEN LOOP
       GOTO INC
       END
```

La variable B0 prend ici les 20 valeurs : 5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

qui renvoient à des sons de différente hauteur (en *crescendo*).

Programme 14

Lire l'état de 5 interrupteurs placés sur le Port A et visualiser l'état de chacun d'eux sur une rangée de 5 LED reliées au Port B.

```
SYMBOL      PORTA = 5
SYMBOL      TRISA = $85
            POKE TRISA, 255
LOOP :      PEEK PORTA, B0
            IF Bit0 = 1 THEN ZERO
            IF Bit1 = 1 THEN UN
            IF Bit2 = 1 THEN DEUX
            IF Bit3 = 1 THEN TROIS
            IF Bit4 = 1 THEN QUATRE
            GOTO LOOP
ZERO :      HIGH 0
            GOTO LOOP
UN :        HIGH 1
            GOTO LOOP
DEUX :      HIGH 2
            GOTO LOOP
TROIS :     HIGH 3
            GOTO LOOP
QUATRE :    HIGH 4
            GOTO LOOP
            END
```

Après les initialisations, et après avoir mémorisé le Port A dans la variable B0, on teste chaque bit qui – s'il est à 1 – renvoie à un sous programme spécifique qui allume la LED correspondante.

Programme 15

Lire l'état de 5 interrupteurs placés sur le Port A et dire lequel d'eux a été fermé, moyennant l'émission de bips sur un haut-parleur placé sur RB7 : un seul bip pour informer qu'il s'agit de l'interrupteur n° 1, deux bips pour le n° 2, et ainsi de suite.. 5 bips pour l'interrupteur n° 5.

```
SYMBOL      PORTA = 5
SYMBOL      TRISA = $85
            POKE TRISA, 255
LOOP :      PEEK PORTA, B0
            IF Bit0 = 1 THEN 1_bip
            IF Bit1 = 1 THEN 2_bips
            IF Bit2 = 1 THEN 3_bips
            IF Bit3 = 1 THEN 4_bips
            IF Bit4 = 1 THEN 5_bips
            GOTO LOOP
1_bip :     SOUND 7, (100, 50)
            GOTO LOOP
2_bips :    SOUND 7, (100, 50, 100, 50)
            GOTO LOOP
3_bips :    SOUND 7, (100, 50, 100, 50, 100, 50)
            GOTO LOOP
4_bips :    SOUND 7, (100, 50, 100, 50, 100, 50, 100, 50)
            GOTO LOOP
5_bips :    SOUND 7, (100, 50, 100, 50, 100, 50, 100, 50, 100, 50)
            GOTO LOOP
            END
```

Après les initialisations, et après avoir mémorisé le Port A dans la variable B0, on teste chaque bit qui – s'il est à 1 – renvoie à un sous programme spécifique qui émet le nombre de bips correspondant.
Les bips ont tous la même hauteur et la même durée (100, 50).
100 correspond à la hauteur de la note. 50 correspond à sa durée.

Programme 16

Transmettre en mode série, à 2400 Bauds, via la pin RB6, un code de 4 bits issu de 4 interrupteurs placés sur le Port A.

```
SYMBOL      PORTA = 5
SYMBOL      TRISA = $85
            POKE TRISA, 255          (Port A en entrée)
LOOP :      PEEK PORTA, B0           (Mémorise le Port A dans B0)
            PAUSE 200
            SERout , T2400, (B0)
            GOTO LOOP
            END
```

- L'instruction PAUSE 200 est nécessaire ici, car le code provient d'interrupteurs (pouvant générer des rebonds).
- L'instruction SERout 6, T2400, (B0) envoie sur la pin 6 du Port B le code stocké dans la variable B0, aux normes T2400 (2400 Bauds, mode TTL vrai).

Programme 17

Recevoir en mode série, à 2400 Bauds, un code entrant sur la pin RB6. Stocker le code reçu dans la variable B0.

```
LOOP :          SERin 6, N2400, B0
                IF B0 > 0 THEN CODE_OK
                GOTO LOOP
CODE_OK         POKE PORTA, B0
                END
```

Le code reçu est stocké dans la variable B0.
L'instruction IF B0 > 0...... sert à détecter l'arrivée d'un code.
Si un code autre que 0 est détecté, alors on le stocke dans B0.
En l'absence de code, on continue à guetter.